

Seminar 3

Data Storage Paradigms, IV1351

Ermia Ghaffari
ermia@kth.se

2023-12-04

1 Introduction

After designing a database for the Soundgood music school and inserting data into the database, it is time to write some queries in SQL. These OLAP queries and views are written for analytical purposes. For instance, in one of the tasks, a query is written to see how many students in school have zero, one or two siblings. After writing queries for different analytical purposes on the database, one of the queries will be explained using "EXPLAIN ANALYZE" which is available in PostgreSQL. At the end, there would be a higher grade part which is about creating a historical database. This database is a denormalized database that keeps track of all lessons taken by all the students with their price, type, student name and email, type of instrument used, and genre of the lesson if it is an ensemble. This task has a marketing purpose.

This task is done with the collaboration of Daniel Ibrahim and Esra Salman.

2 Literature Study

Before getting started with this assignment, chapters 6 and 7 of the book "Fundamentals of Database Systems", were reviewed. The most important part of getting started with the assignment was to understand SQL, and how to use different statements in SQL to achieve the desired outcome of different tasks. Basic knowledge of how databases function in general is also required for the mandatory part of the assignment. It should also be noted that understanding relational algebra by looking at Paris Carbone's on Canvas helped us a lot to get started with understanding SQL queries.

After looking at the lecture on relational algebra and reading the book, a repetition of Paris Carbone's video on SQL was performed. After getting familiar with writing SQL queries, it was time to learn a bit about denormalization and how historical databases function. This was done by researching online and watching some videos.

To summarize, historical databases are used to collect records of past events to analyze them and see trends in them, and denormalization is when merging multiple tables. This

might cause redundancy of data in the database. When using denormalization, complex joins and queries will not be required, and all the data be gathered in one place. Thus, it might lead to system performance in some scenarios.

3 Method

This part will explain the methodology of writing queries in SQL. As mentioned before, all the queries were implemented in a database management called PostgreSQL. In this assignment both the shell of Postgres and PgAdmin were used to run the queries. All the created tables and generated data from the previous assignment were checked before starting to write queries. This was done to avoid any issues when writing the queries. The next step is to decide when to use a view or a materialised view. After discussing with the group members and talking to assistants during the help session, I have decided to use the view on task four. This will be explained more in the discussion part.

Afterwards, queries for different analytical purposes are written. Since the main database with the generated data had too much data, I was forced to create another database and generate a smaller set of data to check the queries gave the desired outcome.

4 Result

The image in the next page shows the logical-physical model to be able to evaluate the queries in the following pages based on this picture. In the following pages, there will be a visual representation of different queries with pictures that represent the outcome of those queries performed on the database. Furthermore, there would be a part at the end of this section, the result, that represents how all the data is transformed from the main database to the historical database.

Afterwards, a query is written to get the desired data from the imported data of the main database into another table in the historical database. This process is fully described in section 4.2. There would be a picture that represents the outcome of recording all the data needed in the historical database.

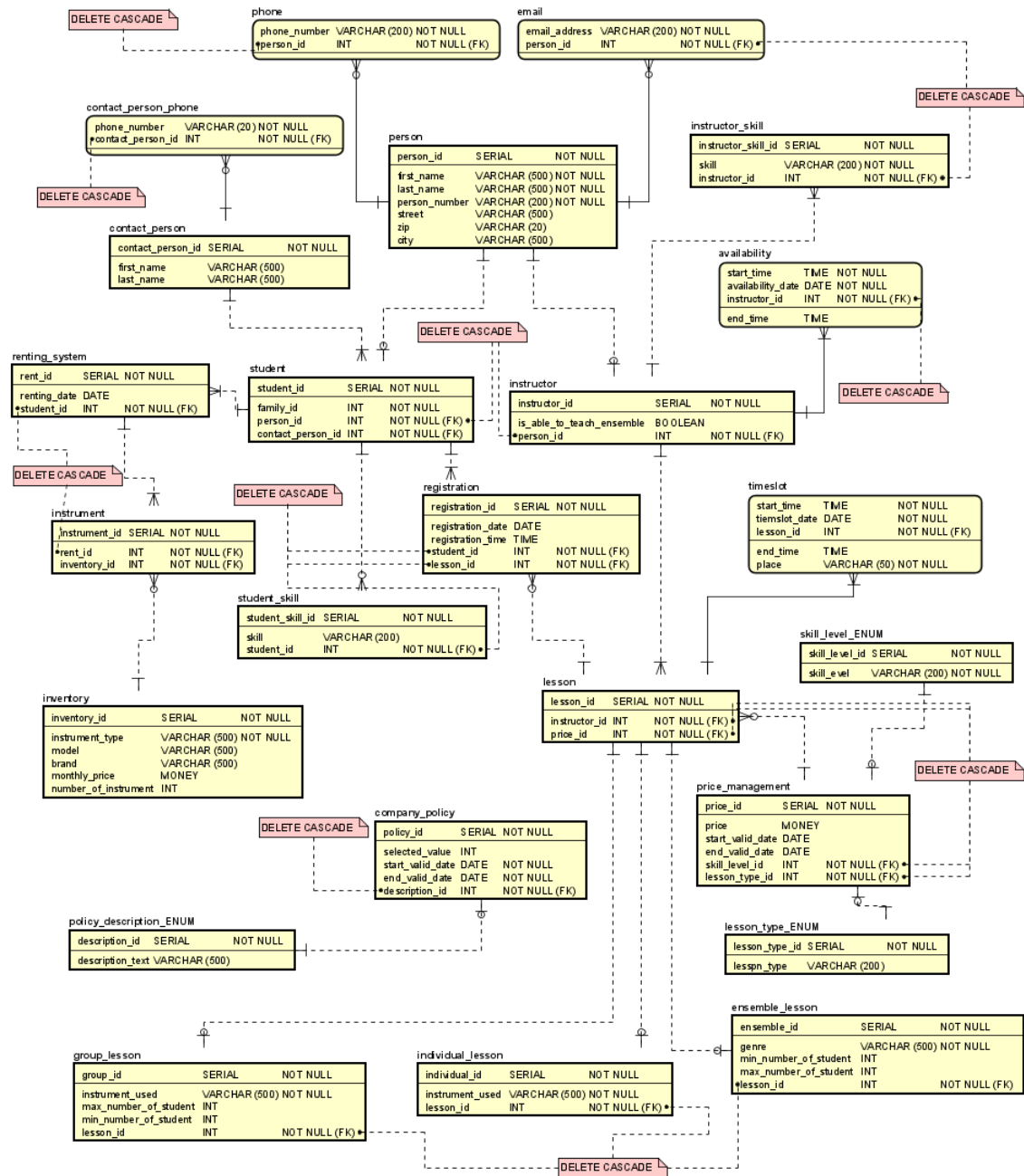


Figure 1: This picture represents the physical model from Sem 2

4.1 Queries

4.1.1 Task 1:

The query below represents how to get the number of lessons per month in a given year. Furthermore, the query should show how the number of individual, group, and ensemble lessons. As seen in Figure 2, the total number of lessons is the addition of all three types of lessons. As shown in Figure 2, the result also shows the month name, and number of individual, group and ensemble lessons. This is done by joining "timeslot," "lesson," "individual-lesson," "group-lesson," and "ensemble-lesson" tables. The WHERE clause filters data for the current year, and that can be changed by the person who writes the query to get a specific year. Thus, the specified years are up to the person to choose. In this case, we have decided to use 2023. The GROUP BY clause groups results by month, and the ORDER BY clause orders them chronologically. As seen at the end, the Order By will be used to display the result based on the moth from Jan-Dec in order.

```
1  SELECT
2      TO_CHAR(T.timeslot_date, 'Mon') AS month,
3      COUNT(IL.lesson_id) AS individual_lesson,
4      COUNT(GL.lesson_id) AS group_lesson,
5      COUNT(EL.lesson_id) AS ensemble_lesson,
6      COUNT(IL.lesson_id) + COUNT(GL.lesson_id)
7          + COUNT(EL.lesson_id) AS total,
8      EXTRACT(YEAR FROM CURRENT_DATE) AS current_year
9  FROM
10     timeslot AS T
11  JOIN lesson ON
12     T.lesson_id = lesson.lesson_id
13  LEFT JOIN individual_lesson AS IL ON
14     IL.lesson_id = lesson.lesson_id
15  LEFT JOIN group_lesson AS GL ON
16     GL.lesson_id = lesson.lesson_id
17  LEFT JOIN ensemble_lesson AS EL ON
18     EL.lesson_id = lesson.lesson_id
19  WHERE
20     EXTRACT(YEAR FROM T.timeslot_date) = EXTRACT(YEAR FROM CURRENT_DATE)
21  GROUP BY
22     TO_CHAR(T.timeslot_date, 'Mon')
23  ORDER BY
24     CASE TO_CHAR(T.timeslot_date, 'Mon')
25         WHEN 'Jan' THEN 1
26         WHEN 'Feb' THEN 2
27         WHEN 'Mar' THEN 3
28         WHEN 'Apr' THEN 4
29         WHEN 'May' THEN 5
```

```

30      WHEN 'Jun' THEN 6
31      WHEN 'Jul' THEN 7
32      WHEN 'Aug' THEN 8
33      WHEN 'Sep' THEN 9
34      WHEN 'Oct' THEN 10
35      WHEN 'Nov' THEN 11
36      WHEN 'Dec' THEN 12
37  END ;

```

	month text	individual_lesson bigint	group_lesson bigint	ensemble_lesson bigint	total bigint	current_year numeric
1	Jan	17	25	12	54	2023
2	Feb	3	17	3	23	2023
3	Mar	36	43	33	112	2023
4	Apr	40	30	17	87	2023
5	May	15	51	3	69	2023
6	Jun	19	24	26	69	2023
7	Jul	8	26	3	37	2023
8	Aug	42	29	9	80	2023
9	Sep	22	50	15	87	2023
10	Oct	9	31	6	46	2023
11	Nov	28	7	15	50	2023
12	Dec	19	15	44	78	2023

Figure 2: This picture represents the outcome of the above query

4.1.2 Task 2:

The purpose of this task is to see how many students in Soundgood music school have zero, one or two siblings. The query for this task is shown below. In this task, the students who have the same family are counted to be siblings. We start by calling the table, family-count, and we will count the number of students with the same family-id. Then, we group the results by their family-id. Then we filter results to include families with only one, two or three children. in other words, families with more students will not be displayed.

the main query will operate from the family-counts. Number of siblings will be calculated by using no-of-siblings minus one. It then calculates the total number of students (number-of-students) based on the count of siblings. the order by at the end will ensure that the output will be in order.

```

1  WITH family_counts AS (
2  SELECT family_id, COUNT(*) AS sibling_count

```

```

3 FROM student
4 GROUP BY family_id
5 HAVING COUNT(*) IN (1, 2, 3)
6 )
7
8 SELECT
9     sibling_count-1 AS no_of_siblings ,
10     SUM(CASE WHEN sibling_count = 1 THEN 1 ELSE 0 END) +
11     SUM(CASE WHEN sibling_count = 2 THEN 1 ELSE 0 END) * 2+
12     SUM(CASE WHEN sibling_count = 3 THEN 1 ELSE 0 END) * 3
13     AS number_of_students
14 FROM family_counts
15 GROUP BY sibling_count
16 ORDER BY sibling_count;

```



	no_of_siblings bigint 	number_of_students bigint 
1	0	154
2	1	142
3	2	87

Figure 3: This picture represents the outcome of the above query

4.1.3 Task 3:

The purpose of this task is to list all the instructors who have given more than a specified number of lessons. The result is shown in Figure 4. In this task, we connect data from the lesson, time slot, and instructor tables by matching the instructor ID. Afterwards, it narrows down the results to only include lessons from the last month and counts them.

After getting this data, the instructor ID is used to get more information like the first name and last name from other sources. These details are shown together with the count of lessons given. Figure 4 presents the outcomes of this query on the experiment's database.

```

1 WITH lesson_count AS (
2     SELECT
3         I.instructor_id,
4         COUNT(*) AS given_lesson_count
5     FROM
6         timeslot AS TS
7     JOIN
8         lesson ON TS.lesson_id = lesson.lesson_id

```

```

9      JOIN
10      instructor AS I ON lesson.instructor_id = I.instructor_id
11  WHERE
12      TS.timeslot_date >= DATE_TRUNC('MONTH',
13      CURRENT_DATE - INTERVAL '1_month')
14      AND TS.timeslot_date < DATE_TRUNC('MONTH', CURRENT_DATE)
15  GROUP BY
16      I.instructor_id
17  )
18
19  SELECT
20      I.instructor_id,
21      P.first_name,
22      P.last_name,
23      LC.given_lesson_count
24  FROM
25      instructor AS I
26  JOIN
27      person AS P ON I.person_id = P.person_id
28  JOIN
29      lesson_count AS LC ON I.instructor_id = LC.instructor_id;

```

	instructor_id integer	first_name character varying (500)	last_name character varying (500)	given_lesson_count bigint
1	3	Kiara	Delaney	1
2	17	Medge	Lynch	2
3	35	Claudia	Mcdowell	1
4	42	Maisie	Miller	1
5	45	Kelsey	Simmons	1
6	51	Xenos	Schwartz	1
7	56	Wylie	Franks	1
8	61	Baker	Tucker	1
9	69	Edan	Fitzpatrick	1
10	83	Jane	Mcdonald	1
11	98	Lucius	Gill	1

Figure 4: This picture represents the outcome of the above query

4.1.4 Task 4:

This query for this task is divided into two different parts. The first part is to create a view and the second part is the analytical table query. This query is used to show all the ensemble lessons during the next week. This query should display an analytical table that illustrates if there is no seat, many seats or if there are "1 or 2" seats left for the ensembles. Furthermore, the genre of ensembles and the number of students that can participate in the ensembles are shown.

The view is created with the name "my-ensemble-count-view", and it selects information about ensemble lessons such as the maximum and minimum number of students that can participate, the genre and the date that these ensembles are given. Furthermore, it also selects the count of students registered for a given ensemble. As seen in the query, joins are made between the lesson, ensemble-lesson, registration, and timeslot tables. Afterwards, a filtration occurs that filters the lesson during the next week, and as it can be seen in Figure 5, The result is represented by lesson ID, day of the week, maximum and minimum student counts, and genre.

The analytical query table down there is the second part of the query code. This part uses the created view. Then, it selects specific columns (day-of-week, genre, number of students) from the view. At the end, there is a case statement that will make different categories like no seat, many seats and "1 or 2 seats". This is done by comparing the number of students enrolled with the maximum allowed students in a specific ensemble.

```

1  --The view:
2  CREATE VIEW my_ensemble_count_view AS
3  SELECT
4      ensemble_lesson.max_number_of_student AS max,
5      ensemble_lesson.min_number_of_student AS min,
6      ensemble_lesson.genre AS genre,
7      TO_CHAR(timeslot.timeslot_date, 'Dy') AS day_of_week,
8      COUNT(*) AS number_of_students
9  FROM
10     lesson AS L
11  JOIN
12     ensemble_lesson ON L.lesson_id = ensemble_lesson.lesson_id
13  JOIN
14     registration AS R ON R.lesson_id = L.lesson_id
15  JOIN
16     timeslot ON timeslot.lesson_id = R.lesson_id
17  WHERE
18     timeslot.timeslot_date >= CURRENT_DATE
19     AND timeslot.timeslot_date < CURRENT_DATE + INTERVAL '1 week'
20  GROUP BY
21     L.lesson_id, day_of_week, max, min, genre;
22  --the query to use to get the analytical table:

```



```

23 SELECT
24     day_of_week ,
25     genre ,
26     CASE
27         WHEN number_of_students > max THEN 'No_Seat '
28         WHEN number_of_students < max
29             AND number_of_students > max-2 THEN '1_or_2_seats '
30         ELSE 'many_seats '
31     END AS number_of_free_seats ,
32     number_of_students
33 FROM my_ensemble_count_view;

```

	day_of_week text	genre character varying (500)	number_of_free_seats text	number_of_students bigint
10	Mon	Hip	many seats	11
11	Mon	Dance	many seats	11
12	Sat	Electronic	No Seat	11
13	Sat	Electronic	many seats	11
14	Sat	Hip	many seats	11
15	Sat	Dance	many seats	11
16	Fri	Country	1 or 2 seats	8
17	Fri	Reggae	many seats	8
18	Fri	Music	many seats	8
19	Fri	Electronic	many seats	8
20	Fri	Electronic	many seats	8
21	Fri	Hop	many seats	8
22	Fri	Country	many seats	8
Total rows: 55 of 55		Query complete 00:00:00.076		

Figure 5: This picture represents the outcome of the above query

4.2 Historical Database (Higher grade)

As explained in the introduction, the purpose of this task is marketing purposes. In this task, the goal is to see which lessons each student has taken. To explain briefly, data will be imported from the main database into a new database using the Postgres-fdw (Foreign Data Wrapper) extension. The purpose as mentioned is to create a historical recording table with information about students, lessons, prices, and other related details.

The first step is to create a new database. This database is called "historical" as shown below.

```
1 CREATE DATABASE historical;
```

The next step is to Connect to the new Database. Afterwards, the Postgres-fdw extension if it doesn't already exist is created as shown below.

```
1 CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

Afterwards, a foreign data server named "histserver" is created. This connects to the main database as shown below.

```
1 CREATE SERVER histserver
2 FOREIGN DATA WRAPPER postgres_fdw
3 OPTIONS (dbname 'new_sound', host 'localhost', port '5433');
```

Afterwards, the creation of user mapping is done for the current user to be able to connect to the foreign data server using the PostgreSQL user. This is also shown below.

```
1 CREATE USER MAPPING FOR current_user
2 SERVER histserver
3 OPTIONS (user 'postgres', password 'xxxx');
```

The next part is to Create a schema named historical-schema within the historical database and import the public schema from the histserver into the historical-schema. This is shown below.

```
1 CREATE SCHEMA historical_schema;
2 IMPORT FOREIGN SCHEMA public FROM SERVER histserver INTO historical_schema;
```

Afterwards, a table called recording will be created in the historical-schema. this table will have record-id, student-name, student-last-name, student-email, lesson-type, genre, instrument-used, and price. the query below explains how the table is created.

```
1 CREATE TABLE historical_schema.recording (
2     record_id SERIAL PRIMARY KEY,
3     student_name VARCHAR(255),
4     student_last_name VARCHAR(255),
5     student_email VARCHAR(255),
6     lesson_type VARCHAR(255),
7     genre VARCHAR(255),
8     instrument_used VARCHAR(255),
9     price NUMERIC
```

10);

The below query shows how to insert data into a table called a recording table. This is done by joining multiple tables together as can be shown in the query. The data is related to student recordings, including names, emails, lesson types, genres, instruments used, and prices. The query for this part is shown below.

```
1  SELECT
2      person.first_name AS student_name,
3      person.last_name AS student_last_name,
4      email.email_address AS student_email,
5      lesson_type_ENUM.lesson_type AS lesson_type,
6      ensemble_lesson.genre AS genre,
7      COALESCE(individual_lesson.instrument_used,
8
9      group_lesson.instrument_used) AS instrument_used,
10     price_management.price AS price
11 FROM
12     historical_schema.student
13     LEFT JOIN historical_schema.person ON
14
15     historical_schema.student.person_id =
16
17     historical_schema.person.person_id
18
19     JOIN historical_schema.email ON
20
21     historical_schema.email.person_id =
22
23     historical_schema.person.person_id
24
25     JOIN historical_schema.registration ON
26
27     historical_schema.student.student_id =
28
29     historical_schema.registration.student_id
30
31     JOIN historical_schema.lesson ON
32
33     historical_schema.lesson.lesson_id =
34
35     historical_schema.registration.lesson_id
36
37     JOIN historical_schema.price_management ON
38
```

```
39     historical_schema.price_management.price_id =
40
41     historical_schema.lesson.price_id
42
43     JOIN historical_schema.lesson_type_ENUM ON
44
45     historical_schema.price_management.lesson_type_id =
46
47     historical_schema.lesson_type_ENUM.lesson_type_id
48
49     LEFT JOIN historical_schema.ensemble_lesson ON
50
51     historical_schema.lesson.lesson_id =
52
53     historical_schema.ensemble_lesson.lesson_id
54
55     LEFT JOIN historical_schema.group_lesson ON
56
57     historical_schema.lesson.lesson_id =
58
59     historical_schema.group_lesson.lesson_id
60
61     LEFT JOIN historical_schema.individual_lesson ON
62
63     historical_schema.lesson.lesson_id =
64
65     historical_schema.individual_lesson.lesson_id;
```

The query shown before is used to display the analytical data with the needed columns to display.

```
1  SELECT
2      recording.student_name ,
3          recording.student_last_name ,
4          recording.student_email ,
5      recording.lesson_type ,
6          recording.price ,
7      recording.genre ,
8      recording.instrument_used
9
10 FROM
11     historical_schema.recording recording;
```

The image below shows the outcome of the above queries. It should be noted that this table is longer than what we see here.

	student_last_name character varying (255)	student_email character varying (255)	lesson_type character varying (255)	genre character varying (255)	instrument_used character varying (255)	price numeric
1	Hahn	volutpat.nulla@aol.couk	Group	[null]	bass	600.00
2	Hahn	volutpat.nulla@aol.couk	Group	[null]	Clarinet	600.00
3	Hahn	volutpat.nulla@aol.couk	Group	[null]	Flute	600.00
4	Stafford	molestie.orci@aol.org	Individual	[null]	Drum	500.00
5	Stafford	molestie.orci@aol.org	Individual	[null]	French	500.00
6	Stafford	molestie.orci@aol.org	Individual	[null]	Cello	500.00
7	Hinton	metus.eu@hotmail.couk	Individual	[null]	Drum	500.00
8	Hinton	metus.eu@hotmail.couk	Individual	[null]	French	500.00
9	Hinton	metus.eu@hotmail.couk	Individual	[null]	Cello	500.00
10	Cummings	sem@protonmail.org	Individual	[null]	Drum	500.00
11	Cummings	sem@protonmail.org	Individual	[null]	French	500.00
12	Cummings	sem@protonmail.org	Individual	[null]	Cello	500.00
13	Barrera	imperdiet.dictum.magna@outlook.net	Individual	[null]	Drum	500.00
14	Barrera	imperdiet.dictum.magna@outlook.net	Individual	[null]	French	500.00
15	Barrera	imperdiet.dictum.magna@outlook.net	Individual	[null]	Cello	500.00
16	Johnson	metus.facilisis@protonmail.couk	Individual	[null]	Drum	500.00
17	Johnson	metus.facilisis@protonmail.couk	Individual	[null]	French	500.00
18	Johnson	metus.facilisis@protonmail.couk	Individual	[null]	Cello	500.00
19	Spencer	eu.neque@icloud.couk	Individual	[null]	Drum	500.00
20	Spencer	eu.neque@icloud.couk	Individual	[null]	French	500.00
21	Spencer	eu.neque@icloud.couk	Individual	[null]	Cello	500.00
22	Byers	nisi@icloud.ca	Individual	[null]	Drum	500.00
23	Byers	nisi@icloud.ca	Individual	[null]	French	500.00
24	Byers	nisi@icloud.ca	Individual	[null]	Cello	500.00
25	Higgins	duis.mi@outlook.com	Individual	[null]	Drum	500.00
Total rows: 1000 of 1606 Query complete 00:00:00.048 Ln 12, Col 1						

Figure 6: This picture represents the outcome of the above queries

5 discussion

To discuss more about the queries written in the result section, there would be more explanation on "why a view is used?", "changing the database to simplify the queries", "usage of subqueries and subqueries that are using data from outer query", "Having unnecessarily long and complicated queries", and doing "Explain Analyze" on the first query. Furthermore, the advantages and disadvantages of using denormalisation are explained at the end of the report.

5.1 Views

As shown in task four, views have been used. Views represent data without needing to store it somewhere in the database. In other words, the view is a virtual table that is created using the SELECT query. In this task, views are used when they are needed. For instance, in task four, the query was a bit complex. Then, the group decided to use a view to make it simpler when someone needs to run that query. The query before using view is displayed below.

```

1 WITH ensemble_count AS (
2 SELECT
3     ensemble_lesson.max_number_of_student AS max,
4     ensemble_lesson.min_number_of_student AS min,

```

```
5      ensemble_lesson.genre AS genre,
6  TO_CHAR(timeslot.timeslot_date, 'Dy') AS day_of_week,
7      COUNT(*) AS number_of_students
8  FROM
9      lesson AS L
10 JOIN
11     ensemble_lesson ON L.lesson_id = ensemble_lesson.lesson_id
12 JOIN
13     registration AS R ON R.lesson_id = L.lesson_id
14 JOIN
15     timeslot ON timeslot.lesson_id = R.lesson_id
16 WHERE
17     timeslot.timeslot_date >= CURRENT_DATE
18     AND timeslot.timeslot_date < CURRENT_DATE + INTERVAL '1 week'
19 GROUP BY
20     L.lesson_id, day_of_week, max, min, genre)
21
22 SELECT
23     day_of_week,
24     genre,
25     CASE
26     WHEN number_of_students > max THEN 'No_Seate'
27     WHEN number_of_students < max
28
29         AND number_of_students > max-2 THEN '1_or_2_seats'
30     ELSE 'many_seats'
31     END AS number_of_free_seats,
32     number_of_students
33 FROM ensemble_count;
```

if looking at the result section, after creating the view a person needs to run the query below only to get the analytical table.

```
1  SELECT
2      day_of_week,
3      genre,
4      CASE
5          WHEN number_of_students > max THEN 'No_Seate'
6          WHEN number_of_students < max
7              AND number_of_students > max-2 THEN '1_or_2_seats'
8          ELSE 'many_seats'
9      END AS number_of_free_seats,
10     number_of_students
11 FROM my_ensemble_count_view;
```

Thus, as seen using views will help with simplifying complex queries. Thus, it can also be used to hide complexity. Furthermore, views can be used on queries that are going to run multiple times during a short time. Furthermore, security is another provided feature of using view. Additionally, views provide an abstraction layer on the table. Performance can be improved sometimes when using views. In this case, when explaining the analysis of the query with a view and without, the performance and the execution time do not change as much.

5.2 Optimization and Compromise

Our group made important decisions before starting to write the SQL queries. For instance, when working the historical databases, we had two different options, one to export all the data manually and import it to the new database or use the code snippets that Leif has provided on Piazza to automatically do it. When creating the historical database we could make the model from the previous task simpler. Changing the model would make the query on task four a bit easier since there were multiple joins in the task. The group decided to look for a solution to reduce the number of joins. For instance, combining individual lessons, group lessons, and ensemble lessons into one lesson entity was an option for the group. However, this would result in redundancy and having too many duplicates.

We also had some ideas to merge email, person, and student entities to quickly get access to email, name, and lesson information without needing extra steps to select attributes. The issue with this idea is as same as the issue above.

There were also some issues with the choice of data format in task 1. We used only three letters for months. That resulted in having some challenges when organising the dates. Thus, we had to manually arrange months. This issue was not directly connected to the model, however, the generated data had some issues. Hence, some changes were made due to the need.

To summarize, the group did not change too many things from the previous tasks. This is because we want to keep the well-organized design from the previous task. In other words, the group did not choose the easier way to write the query, and even though the queries were harder to write without changes, the group decided to choose that oath.

5.3 subqueries and having unnecessarily long and complicated queries

There are no correlated subqueries in the queries shown in the result section. correlated subqueries are the subqueries that use values from the outer query. these queries could be slow because these queries are evaluated for each row that is getting processed in the outer query. In the queries shown in the result, there is no subquery such as WITH clauses and the view definition that depends on values from the outer queries.

It should be also noted that there are no unnecessarily long and complicated queries used in this task.

5.4 advantages and disadvantages of denormalisation

Denormalization is already explained in the literature study part of the essay. As mentioned before, denormalization is when merging multiple tables. This might cause redundancy of data in the database. When using denormalization, complex joins and queries will not be required, and all the data be gathered in one place. Thus, it might lead to system performance in some scenarios. Additionally, there would be more storage required when having denormalised data and this can be seen as a huge disadvantage. As a result, there would be some maintainability challenges.

5.5 Explain Analyze

Explain Analyze is a feature in PostgreSQL. In this task, we used the first query to run the explained analysis. here is the "Planning Time: 1.530 ms" and the "Execution Time: 11.007 ms". When analysing how the query is performed, a few things slow down the execution. When we arrange the results in order based on the month from Jan-Dec, it takes most of the time, and this is because too much data is involved in this process. Second, when we try to combine information from different tables, especially when dealing with the 'ensemble-lesson,' 'group-lesson,' and 'individual-lesson' tables and connecting them with the 'lesson' table, it's not as quick. this happens three times. It happens once for each of these smaller tables.

To simplify, imagine you have different lists of information about lessons, like those involving ensembles, groups, and individuals. Now, to make sense of all these lists, we need to connect them using the 'lesson' table. This connecting process takes a bit more time, especially when we're dealing with a lot of information.

Towards the end of the plan, there's a step where we filter out some data based on what we found earlier. This part is quicker because we've narrowed down our focus to a smaller set of information.

In simpler terms, the main challenges in making this query run smoothly come from the steps of organizing the results and connecting data from various tables. Even though these steps take some time, they are necessary to get the final results we're looking for.

6 Link to Github

<https://github.com/ermia1230/IV1351.git>